Calling a Web API From a .NET Client (C#)

2014-1-20 • 6 min to read • Contributors

In this article

by Mike Wasson

Download Completed Project

This tutorial shows how to call a web API from a .NET application, using System.Net.Http.HttpClient.

In this tutorial, we will write an client application that consumes the following web API.

Action	HTTP method	Relative URI
Get a product by ID	GET	/api/products/*id*
Create a new product	POST	/api/products
Update a product	PUT	/api/products/*id*
Delete a product	DELETE	/api/products/*id*

Note

To learn how to implement this API on the server, using ASP.NET Web API, see Creating a Web API that Supports CRUD Operations.

For simplicity, the client application in this tutorial is a Windows console application. **HttpClient** is also supported for Windows Phone and Windows Store apps. For more information, see Writing Web API Client Code for Multiple Platforms Using Portable Libraries

Create the Console Application

In Visual Studio, create a new Windows console application and paste in the following code.

C# Copy

```
using System;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;

namespace HttpClientSample
{
    class Program
    {

        static void Main()
        {
            RunAsync().Wait();
        }

        static async Task RunAsync()
        {
            Console.ReadLine();
        }
    }
}
```

This code provides the skeleton for the application. The `Main` `RunAsync` method and blocks until it completes. The reason for this approach is that most **HttpClient** methods are async, because they perform network I/O. All of the async tasks will be done inside `RunAsync`. In a console application, it's OK to block the main thread inside of `Main`. In a GUI application, you should never block the UI thread.

Install the Web API Client Libraries

Use NuGet Package Manager to install the Web API Client Libraries package.

From the **Tools** menu, select **Library Package Manager**, then select **Package Manager Console**. In the Package Manager Console window, type the following command:

console Copy

```
Install-Package Microsoft.AspNet.WebApi.Client
```

Add a Model Class

Add the following class to the application:

C# Copy

```
namespace HttpClientSample
{
    public class Product
    {
        public string Id { get; set; }
        public string Name { get; set; }
        public decimal Price { get; set; }
        public string Category { get; set; }
    }
}
```

This class matches the data model used by the web API. We can use **HttpClient** to read a `Product` instance from an HTTP response, without having to write a lot of deserialization code.

Create and Initialize HttpClient

Add a static **HttpClient** property to the `Program` class.

C# Copy

```
class Program
{
    // New code:
    static HttpClient client = new HttpClient();

}
```

Note

HttpClient is intended to be instantiated once and re-used throughout the life of an application. 2
Especially in server applications, creating a new **HttpClient** instance for every request will exhaust the number of sockets available under heavy loads. This will result in **SocketException** errors.

To initialize the **HttpClient** instance, add the following code to the `RunAsync` method:

C# Copy

```
static async Task RunAsync()
{
    // New code:
    client.BaseAddress = new Uri("http://localhost:55268/");
    client.DefaultRequestHeaders.Accept.Clear();
    client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("applic
    Console.ReadLine();
}
```

This code sets the base URI for HTTP requests, and sets the Accept header to "application/json", which tells the server to send data in JSON format.

Sending a GET request to retrieve a resource

The following code sends a GET request for a product:

C# Copy

```
static async Task<Product> GetProductAsync(string path)
{
    Product product = null;
    HttpResponseMessage response = await client.GetAsync(path);
    if (response.IsSuccessStatusCode)
    {
        product = await response.Content.ReadAsAsync<Product>();
    }
    return product;
}
```

The **GetAsync** method sends the HTTP GET request. The method is asynchronous, because it performs network I/O. When the method completes, it returns an **HttpResponseMessage** that contains the HTTP response. If the status code in the response is a success code, the response body contains the JSON representation of a product. Call **ReadAsAsync** to deserialize the JSON payload to a `Product` instance. The **ReadAsync** method is asynchronous because the response body can be arbitrarily large.

HttpClient does not throw an exception when the HTTP response contains an error code. Instead, the **IsSuccessStatusCode** property is **false** if the status is an error code. If you prefer to treat HTTP error codes as exceptions, call HttpResponseMessage.EnsureSuccessStatusCode on the response object. This method throws an exception if the status code falls outside the range 200–299. Note that **HttpClient** can throw exceptions for other reasons — for example, if the request times out.

Using Media-Type Formatters to Deserialize

When **ReadAsAsync** is called with no parameters, it uses a default set of *media formatters* to read the response body. The default formatters support JSON, XML, and Form-url-encoded data.

Instead of using the default formatters, you can provide a list of formatters to the **ReadAsync** method, which is useful if you have a custom media-type formatter:

C# Copy

```
var formatters = new List<MediaTypeFormatter>() {
    new MyCustomFormatter(),
    new JsonMediaTypeFormatter(),
    new XmlMediaTypeFormatter()
};
resp.Content.ReadAsAsync<IEnumerable<Product>>(formatters);
```

For more information, see Media Formatters in ASP.NET Web API 2

Sending a POST Request to Create a Resource

The following code sends a POST request that contains a `Product` instance in JSON format:

C# Copy

```
static async Task<Uri> CreateProductAsync(Product product)
{
    HttpResponseMessage response = await client.PostAsJsonAsync("api/products", product
    response.EnsureSuccessStatusCode();

    // Return the URI of the created resource.
    return response.Headers.Location;
}
```

The **PostAsJsonAsync** method serializes an object to JSON and then sends the JSON payload in a POST request. If the request succeeds, it should return a 201 (Created) response, with the URL of the created resources in the Location header.

Sending a PUT Request to Update a Resource

The following code sends a PUT request to update a product.

C# Copy

```
static async Task UpdateProductAsync(Product product)
{
    HttpResponseMessage response = await client.PutAsJsonAsync($"api/products/{product.
    response.EnsureSuccessStatusCode();

    // Deserialize the updated product from the response body.
    product = await response.Content.ReadAsAsync<Product>();
    return product;
}
```

The **PutAsJsonAsync** method works like **PostAsJsonAsync**, except that it sends a PUT request instead of POST.

Sending a DELETE Request to Delete a Resource

The following code sends a DELETE request to delete a product.

C# Copy

```csharp
static async Task DeleteProductAsync(string id)
{
    HttpResponseMessage response = await client.DeleteAsync($"api/products/{id}");
    return response.StatusCode;
}
```

Like GET, a DELETE request does not have a request body, so you don't need to specify JSON or XML format.

Complete Code Example

Here is the complete code for this tutorial. The code is very simple and doesn't include much error handling, but it shows the basic CRUD operations using **HttpClient**.

C# Copy

```csharp
using System;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;

namespace HttpClientSample
{
    public class Product
    {
        public string Id { get; set; }
        public string Name { get; set; }
        public decimal Price { get; set; }
        public string Category { get; set; }
    }

    class Program
    {
        static HttpClient client = new HttpClient();
        static void ShowProduct(Product product)
        {
            Console.WriteLine($"Name: {product.Name}\tPrice: {product.Price}\tCategory:
        }

        static async Task<Uri> CreateProductAsync(Product product)
        {
            HttpResponseMessage response = await client.PostAsJsonAsync("api/products",
            response.EnsureSuccessStatusCode();

            // return URI of the created resource.
            return response.Headers.Location;
        }

        static async Task<Product> GetProductAsync(string path)
        {
            Product product = null;
            HttpResponseMessage response = await client.GetAsync(path);
            if (response.IsSuccessStatusCode)
            {
                product = await response.Content.ReadAsAsync<Product>();
            }
            return product;
        }

        static async Task<Product> UpdateProductAsync(Product product)
        {
            HttpResponseMessage response = await client.PutAsJsonAsync($"api/products/{
            response.EnsureSuccessStatusCode();

            // Deserialize the updated product from the response body.
            product = await response.Content.ReadAsAsync<Product>();
            return product;
        }

        static async Task<HttpStatusCode> DeleteProductAsync(string id)
        {
            HttpResponseMessage response = await client.DeleteAsync($"api/products/{id}
            return response.StatusCode;
        }

        static void Main()
        {
            RunAsync().Wait();
        }

        static async Task RunAsync()
        {
            client.BaseAddress = new Uri("http://localhost:55268/");
            client.DefaultRequestHeaders.Accept.Clear();
            client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue

            try
            {
                // Create a new product
                Product product = new Product { Name = "Gizmo", Price = 100, Category =

                var url = await CreateProductAsync(product);
                Console.WriteLine($"Created at {url}");
                // Get the product
                product = await GetProductAsync(url.PathAndQuery);
                ShowProduct(product);

                // Update the product
                Console.WriteLine("Updating price...");
                product.Price = 80;
                await UpdateProductAsync(product);

                // Get the updated product
                product = await GetProductAsync(url.PathAndQuery);
                ShowProduct(product);

                // Delete the product
                var statusCode = await DeleteProductAsync(product.Id);
                Console.WriteLine($"Deleted (HTTP Status = {(int)statusCode})");

            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }

            Console.ReadLine();
        }

    }
}
```